Software product line engineering for consumer electronics

Hartmann, Herman

Chapter 9. Risk Based Testing for Software Product Line Engineering

This chapter is an extended version of the paper that was published as:
H. Hartmann, F. van der Linden, J. Bosch: Risk based testing for software product line engineering. SPLC 2014: 227-231, DOI 10.1145/2648511.2648535.

Abstract.
The variability of product lines increases over time thereby leading to an increasing effort for testing. Since the available time for test activities is limited an efficiency improvement is needed to ensure that products have sufficient quality.

This paper introduces risk-based testing for software product lines. Our approach is based on risk based testing for single system engineering which is extended with a dimension that captures the percentage of product variants that use a particular development artefact. Based on the risk of development artifacts, the priorities for testing during domain and application engineering are determined. We demonstrate our approach using a case study from an existing product line and discuss the tool support that is needed.

We conclude that the basic ideas behind risk-based testing product lines are intuitive, pragmatic in nature, and provide the means for practitioners for guiding the test effort for different test levels.

9.1 Introduction

Software Product line engineering improves the efficiency of developing variants of software products by using a set of common artifacts [Pohl 2005A]. The amount of test effort as a percentage of the total development effort has dramatically increased [Siewiorek 2004]. The testing costs for product lines are high and effort reduction strategies can significantly improve profitability and productivity [daMotaSilveira 2011]. In most organizations the time for testing is limited and therefore many products cannot be tested thoroughly, resulting in lower product quality, a loss of market share or financial loss [Tassey 2002]. It is therefore essential to focus the test activities to those areas where the biggest risks are.

The development and testing of software product lines is usually separated into domain engineering and application engineering. The main focus of testing during domain engineering is on unit and subsystem testing of the reusable development artifacts while that of testing during application engineering is on integration and system testing [McGregor 2010]. Testing during domain engineering has the limitation that the large combination of variation points and combination of components leads to a combinatorial explosion of possible configuration which, given the limited time that is available for testing

is unfeasible. Furthermore, for many of the components and subsystems of a product line it is not known how many of the derived products use a specific component and what their quality requirements are. As a result it is unclear how the limited test effort should be allocated to the different components.

Risk Based Testing (RBT) is a widely adopted practice in single system engineering [Amland 2000, Alam 2013]. It is used to prioritize test effort to those development artifacts that pose the greatest risk. This risk is based on two dimensions (1) the probability that a function or feature contains defects and (2) the impact that a failure causes. The risks are identified by the different stakeholders and visualized in a risk matrix [Amland 2000, Veenendaal 2014, Schaefer 2014].

The approach described in this paper extends risk based testing to software product line engineering (SPLE) in the following ways:

- For each development artifact it is decided, based on its percentage of use in the set of variants, whether it should be tested during DE or that test should be deferred to AE. The percentage of use forms an additional dimension in RBT.
- The percentage of use of development artifacts and impact when it fails are derived from the features they implement.

To capture the risks and identify the most critical components, this paper uses feature models [Kang 1990] in which each feature is associated with a percentage of the use of a feature as well as the impact of a feature. As an example, Figure 49 shows a diagram of a quantified feature model of a subset of the variability of a digital television.

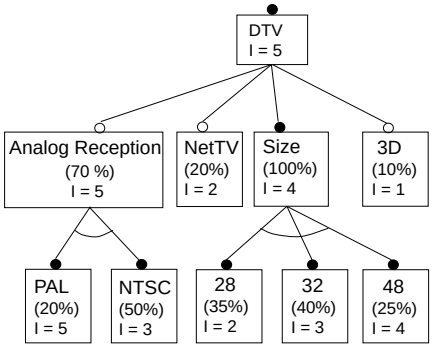

Figure 49 Quantified feature model of a digital television

This diagram shows that 70% of the variants contain Analog Reception of which 20% supports PAL and the other 50% NTSC, etcetera. Furthermore, in this example, PAL is a feature that has high impact on the user or the business (value=5) while 3D isn't (value = 1). This paper shows that these feature models can be used to quantify the use of development artifacts in the different configurations, since development artifacts are linked to features. The risk matrix is derived by including the impact of a feature and the probability of a component causing a failure.

In general a feature model with quantification, which will be denoted as quantified feature models (QFM), has to adhere to consistency rules. In the example in figure 1 we see that the

percentage of PAL and NTSC together are equal to the percentage of their parent feature. These consistency rules can become complex thereby leading to inconsistent QFMs. Therefore this paper also discusses the modeling aspects and the tool support that is needed to support the risk assessment process.

In this paper we answer the following research questions:

- How can risk based testing by applied in Product line Engineering?
- What are the consequences for testing during domain and application engineering?
- What tool support is required?

This paper is structured as follows: Section 9.2 gives a background on product lines testing and risk based testing. Section 9.3 describes risk-based testing for software product lines, while Section 9.4 presents a case study and Section 9.5 discusses tool support. This paper concludes with a comparison with related art, our conclusions and a description of further research.

9.2 Background

This section provides the background on testing of product lines and risk based testing and provides the problem description.

9.2.1 Strategies for product line testing

In software product line engineering tests are executed during domain engineering (DE) and during application engineering (AE) [Pohl 2005A, McGregor 2010]. Usually the testing is separated into different test levels:

- A unit test verifies a component or software unit against its specification.
- An integration test verifies the interaction between integrated software units and components.
- A system test verifies the whole system against the specification of the product.

Sometimes sub systems, or compound component are used thus introducing an additional level of testing.

In dividing the test activities between DE and AE the following strategies are known [Pohl 2005A, daMotaSilveira 2011, Kang 1990]:

Brute force: All test levels and combinations of variation points and components are performed during DE. Given the combinatorial explosion of the amount of features and combination of components this is unfeasible in most situations.

Testing product by product: All test levels are performed during AE. The advantage of this strategy is that only the components and variation points are tested that are actually used. The disadvantages of this strategy are that defects are found late in the development process and, since some components may be used in many products, many tests are redundant as the same defects may be found in different products. This leads to additional effort to correct the defects in the various configurations.

Division of responsibilities: In this strategy the unit tests are performed on the reusable components during DE, while the integration and system test are performed during AE. The

advantage of this strategy is that no additional effort is needed during DE to test the interaction of components. A disadvantage is that still a large, possibly huge, number of tests of reusable components is needed and that failures that are caused by the interaction of components cannot be found early.

Test assets for reuse: In this strategy the test assets, containing variability, are created during DE. During DE the common parts are tested while the application specific parts are tested during AE using the reusable test assets. Compared to the Division of Responsibility strategy, in this strategy the tests can be performed during AE and only for those variation points that are actually used.

Sample application strategy: A few sample applications are selected, based on a selection of components and variation points, and all levels of testing are performed. During AE each application needs to be retested with its specific components and bound variation points. The advantage of this strategy is that it shows that an application can be created from the reusable components and that a selection of the components is also tested on their interfaces between them. A disadvantage of this strategy is that the creation of a sample application causes overhead since this application will not be delivered to customers.

In practice the Division of responsibility is, to our knowledge, the most applied strategy and some case studies are described [McGregor 2010, Ganesan 2007], the Brute force and Testing product by product strategies are not advised [Pohl 2005A] and are not further discussed in this paper.

9.2.2 Defining test techniques and coverage

The test plan defines which test techniques and test coverage are required for each test level [McGregor 2010]. The test techniques may include reviews, static analysis and dynamic testing, such as white- and black box testing. Test coverage is a measure of how thorough each test, for each test level, needs to be executed. This defines what percentage of the code is tested, e.g. the function or statement coverage and what percentage of the requirements are tested. The test coverage determines how many test cases need to be created, which is the most labor intensive part of testing [Pohl 2005A]. The challenge in defining the required test techniques and coverage is to find the right balance between the optimal time-to-market and costs versus the required quality [Veenendaal 2014]. For safety critical systems, such as healthcare equipment, the reliability is essential thus requiring a high coverage, while for consumer products the time-to market is more important and less coverage is needed [Musa 1999]. For instance for some of the components test may include code reviews and dynamic testing with a 90% coverage, while for other components, where less quality is required, only dynamic testing with a 50% coverage is sufficient.

9.2.3 Risk based testing

In many companies the concept of RBT is used to support the planning of test activities for system testing [Amland 2000]. RBT is based on the notion that not everything can be tested and that (only) the most important defects should be found and removed. For areas with high risks the tests are executed earlier, using more comprehensive test techniques and higher

coverage targets are set [Schaefer 2014], while for non-critical areas lower coverage are set [daMotaSilveira 2011].

Usually a matrix visualizes the risks, which consists of two dimensions: The probability dimension captures the likelihood that a function or feature fails and the impact dimensions capture the consequence that a feature or function fails.

The values for probability and impact are determined by the stakeholders in a project, e.g. the product managers, developers and testers. These values are based on expert opinion, rather than rigorous statistics. Through a guided discussion each of the participants scores the factors that determine the values [Veenendaal 2014].

For the probability the following factors are typically used: complex or changed areas, new technology, number of people involved, time pressure and areas that revealed many defects in the past. For the impact typically the following factors are used: cost and consequences of a failure, visibility to the end user and usage frequency [Schaefer 2014].

The values of the individual aspects are added, often using a weighted sum, and this gives the risk matrix, of which an example is shown in Figure 50.

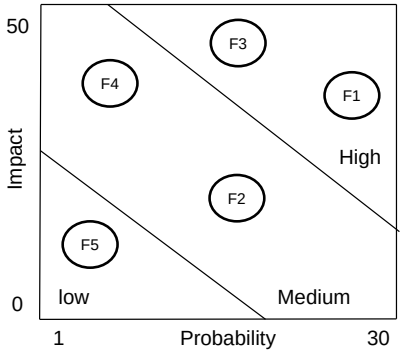

Figure 50 Example of a risk matrix

In this example the features F1 and F3 have a high risk and should be tested intensely, meaning that more comprehensive test techniques should be used and higher coverage targets should be set, while for feature F5 less test effort should be allocated.

9.2.4 Problem description

Given the limited time that is available for test activities it is needed to test those components and variation points that have the biggest impact on the product quality and support the strategies most efficiently. More specifically the following questions need to be answered:

- **Division of responsibilities:** Which components and which variation points should be tested during DE and for which the tests can be deferred to AE?
- **Test assets for reuse:** For which components and variation points should reusable test assets be created during DE?

- **Sample application strategy:** Which components and variation points should be selected for a sample application?

9.3 Applying Risk Based Testing to Product Line Engineering

This section describes how RBT can be applied to product line engineering.

9.3.1 Approach

Our approach is based on the following additions to RBT in single system engineering:

- The risks at system level are translated into risks to individual development artifacts, such as software components, units or sub-systems.
- The development artifacts are identified which pose the largest risk to the product line as a whole.
- A selection is made of which artifacts need to be tested during DE and for which testing can be deferred to AE.

9.3.2 Obtaining the risks at component level

In order to obtain the risk and set priorities for components and subsystems, the risks that are identified at system level have to be allocated to those components and subsystem that implement the features.

Since development artifacts are linked to the features [Pohl 2005A], the quantification of components follows from the quantification of features. Look at the following example, presented in Figure 51. This example is simplified for reasons of clarity. The figure shows a feature model with the components that implement the features, of which part of the links between them are shown.

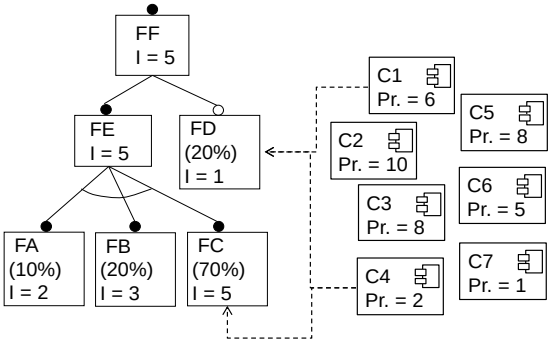

Figure 51 Feature model with linked components

The components are linked to the features, with the following dependency rules:

- **C1 → FD** i.e. Component C1 is present in a configuration when feature FD is present in the configuration,

- **C2 → FA**, **C3 → FB**, **C4 → FC AND FD**, i.e. Component C4 is present when both features FC and FD are present in the configuration.
- **C5 → FC AND ¬FD**, i.e. Component C5 is present when both FC is present but not feature FD.
- **C6 → FC OR FD**, i.e. Component C6 is present when either features FC or FD is present in the configuration**,**
- **C7 → FF** i.e. Component C7 is present in every configuration.

When defining the risks of an individual component, three dimensions are relevant. The first two dimensions are equal to those used in RBT for single system engineering, namely the probability and the impact. The third dimensions originates from Product Line Engineering, namely what percentage of product variants use a specific feature, and therefore component.

The probability of a component: The probability that a component fails is shown in the figure, as attribute of a component. This is defined as a number between 1 and 10.

The impact of a component: The impact of a component is derived from the features it implements and is a number between 1 and 5. Since the impact of a compact depends on the impact of the highest impact of the linked features, we take the maximum value of the features it implements, rather than e.g. an average value. Thus we get: I(C1) = I(FD) = 1, I(C2) = I(FA) = 2, I(C3) = I(FB) = 3, I(C4) = Max(I(FC), I(FD)) = 5, I(C5) = I(FC) = 5, I(C6) = Max(I(FA), I(FD)) = 5, I(C7) = I(FF) = 5.

The percentage of component usage: The percentage for the components, P(C1), …P(C6) follow from the percentages of the features. Here we assume that the percentages of the features are independent from each other. So we get: P (C1) = P(FD) = 20 %, P (C2) = P(FA) = 20 %, P (C3) = P(FB) = 20 %, P (C4) = P(FC AND FD) = 14 %, P (C5) = P(FC AND ¬FD) = 56 %, P (C6) = P(FA OR FD) = 76 %.

Note1: In this example we have used a direct link between a component and a feature. In a more generic case a component, or other development artifact, may be indirectly linked to a feature. For instance a component may depend on the presence of another component. For instance an additional component C8 is present when component C1 or C3 is present. Furthermore a component may contain software units, which presence in a configuration also depends on a variation point.

Note2: As stated in this example it assumed that feature percentages are independent, e.g. the choice for feature FC and FD are independent. However, the choice of one feature may influence the choice for another

In Section 9.5 tool support and quantified feature models are discussed in more detail.

9.3.3 Risk matrix for reusable components

The results of the scores can be presented in a matrix. Since we have three dimensions rather than two, the scores for impact and probability have been combined into one axis.

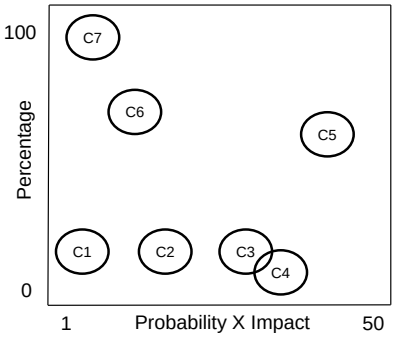

Figure 52 Risk matrix for reusable components

9.3.4 Risk based testing during domain engineering

The risk matrix is the basis for making choices during DE depending on the different strategies:

Division of responsibilities: For this strategy the components and variation points that are used in many products should be tested as part of DE, especially those that have a high probability and implement features with high impact. The components that are used in a small percentage of products and have little probability and impact should not be tested during DE.

Test assets for reuse: Since creating reusable test assets is work intensive a selection can be made. This selection is similar to the division of responsibility strategy.

Sample application strategy: Again here we see that the components and variation points that are used in many products should be included in the sample application, while the others should not be. During the testing of the sample application more test effort should be devoted to the components that implement features with high impact and have high probability of causing a failure.

A matrix such as shown in Figure 52 provides clear guidance. Some examples: Component C7 is used in all products and therefore should be tested during DE although no intensive testing is required as the quality is already considered as sufficient. Component C5 has high risk and is used in the majority of products and should be tested intensively. Component C3 and C4 are not used in many products and therefore testing should be done during AE only, while Components C1 and C2 are hardly used and because they pose little risks do not need to be tested in separation anymore.

The selection of components based on their risks for the product line addresses the disadvantages of the different strategies. For the Division of responsibilities strategy it is avoided that components that are not used heavily are tested, for the Test assets for reuse strategy only a limited set of reusable test artifacts need to be created while for the sample application strategy a combination of components is selected for which the probability is high that they will be used in an actual product. In this way it is avoided that effort is spend during DE which cannot be justified.

9.3.5 Risk based testing during application engineering

During this phase risk based testing can be applied very much in the same way as risk based testing for single system engineering and a risk matrix can be applied to capture the risks of the features for a specific application.

An important input to the risk table, related to the probability dimension, is whether components have already been tested during DE. For instance, those components that have already been tested intensively during DE, and with the variation points used for that application, should not be tested heavily anymore. AE can focus on application specific components and on the interaction between components.

9.3.6 Comparison of risk analysis processes

The process of RBT for product lines differs from that of single system engineering, see e.g. [Veenendaal 2014]. Table 23 shows the differences.

Table 23 Comparison of risk analysis processes

Step	**Single System**	**Product Lines**
Estimate impact of a feature	Yes	Yes
Estimate percentage of a feature in the product line	Not applicable	Yes
Calculate percentage for each component	Not applicable	Yes
Estimate probability per component	Yes	Yes
Visualize the risks in a risk matrix	Yes	Yes
Choose which components should be test during DE and which during AE.	Not applicable	Yes
Prioritize test activities.	Yes	Yes

9.4 Case Study: Philips Healthcare

This section describes a case study of Philips Healthcare that uses RBT as part of their software product line testing.

9.4.1 Introduction to the case study

Philips Healthcare manufactures a wide range of medical imaging equipment, including systems for image acquisition such as X-ray, Magnetic Resonance Imaging (MRI) and Computer Tomography (CT). Philips uses a product line approach for the Philips Medical Workspot (PMW) which is a product line of system used by radiologists to analyze the images that are created by the different medical imaging equipment [Linden 2007]. The

PMW supports storing, retrieving and exchanging of medical images as well as applications that assist the radiologists to determine a diagnosis.

The product line is based on a reference architecture and reusable components, see Figure 53. DE and AE are strictly separated in different organizational units. The platform development group is responsible for creating and testing the components as well as the platform. The product groups, i.e. the groups that create the PMW for the CT, MRI X-ray systems etc., are responsible for creating and testing the end-product and bringing it to the market. The platform for the PMW consists of components that are common for all products as well as optional component and software units that are used by more than one product group. This platform is a semi-final product meaning that the platform can be configured by the product groups. The product groups can add components from the set of reusable components; modify these and add product specific components.

External regulations: Because healthcare equipment requires high quality the approval of legal authorities is needed to sell this equipment. This includes the identification and resolving of risks. For the PMW this means that for features that can cause a safety or security risks additional measures have to be taken, which includes a validation of the test cases. These features include the analyses and reporting of medical images. For other features, such as printing of cleaning up databases less rigorous testing is required.

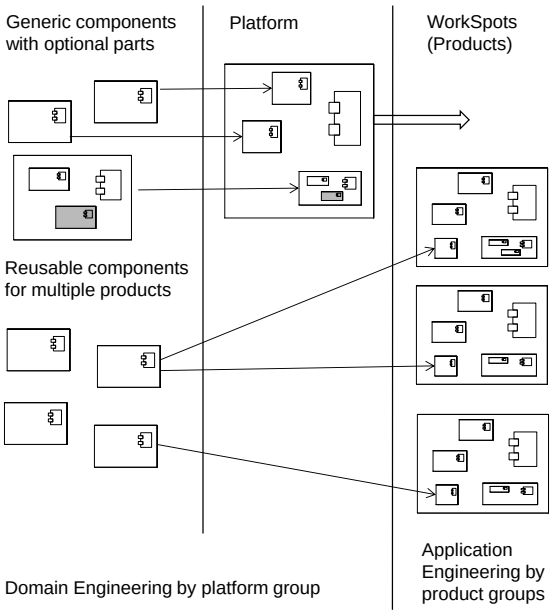

Figure 53 Philips Medical Workspot platform development

9.4.2 Testing and priorities

Testing is performed at three levels: (1) Unit and Component testing (2) Testing of the Platform and (3) Testing of the End Products:

(1) Unit and Component Testing: The test of the reusable component is performed by the DE teams, while the product specific units and components are tested by the product groups. For all components test automation is applied where each component has 100 % requirement coverage and at least 80 % line coverage. For a selected set of components a test script validation procedure is in place which checks the intended use of the test script against its test design and outputs. These components are: (A) Components that implement Safety and Security Features. (B) Components that are used by multiple customers and for which it is decided that the responsibility for the quality of these components lies with the DE team. This means that these internal customers do not separately test these components during AE.

(2) Testing of the platform: The platform tests are executed by the DE Team. This team uses RBT where the process of Veenendaal [Veenendaal 2014] is used and the factors from Schaefer [Schaefer 2014]. The probability dimension is based on 10 factors such as the complexity of the component, whether it is new or implemented with 3rd party components and whether the components contained many defects in the past. The Impact dimension is based on 3 factors: The damage a feature might give; the usage frequency and the commercial importance. An example is given in Figure 54 in which, for reasons of confidentiality, the names of the features have been removed:

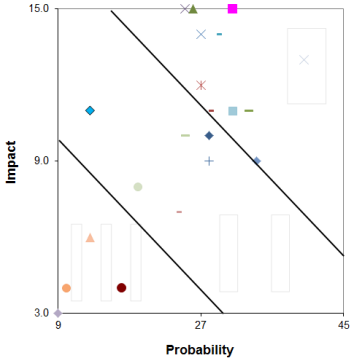

Figure 54 Risk matrix for Philips Medical Workspot platform

In this analysis an exception is made for features that can cause safety or security risks. Even if the analysis of probability and impact shows that the feature is not of high risk, it will be treated as such.

(3) Testing of the end product: The product groups perform testing at system level and testing of product specific components. For system tests the standard method of RBT is applied, but now including the features that have been added by the product team. Furthermore the product teams obtain information from the usage of features at the customer's site and in-house measurement. This information is used as input for the risk matrix [Hartmann 2006, and chapter 8] and for performing reliability and performance testing.

9.4.3 Selection of components for domain engineering

For each component it is identified which of the products use these components and thus the amount, and percentage, of products that use a component is identified. During meetings with the various stakeholders, such as architects and product managers, it is decided for which components the testing should be done by the DE team and which components should become part of the platform. Furthermore the risks are identified for each component. In this way the most critical and most used components are not only tested in isolation but also tested in combination with other components, as part of the platform.

9.4.4 Tool support

The variability is managed hierarchically, using different mechanisms and different solutions at many levels of the hierarchy (see [Linden 2007], Chapter 13). At the top level component selection is the main mechanism. The relation between the features and the components that implement them is a simple 1-to-n relation and therefore no feature and family modelling tools, that represent these relations, are used. In other words, it is clear to the stakeholders which components implement which features and therefore the discussion is about the components and their criticality rather than the criticality of features.

9.4.5 Analysis and summary of this case study

The test activities for this product line is a combination of two strategies:

(1) ***Division of responsibility:*** The DE group is responsible for the testing of the reusable components and has to ensure that these components function properly in the platform. The product groups are responsible for testing application specific components and the end product.

(2) ***Sample application***: The semi-final product serves both as a platform as well as a sample application to test the reusable components and their interaction.

RBT, using a risk table, is applied for the platform as part of DE and during AE by the product groups.

This case study only validates a part of our approach presented in Section 9.3:

- The selection of components to be tested by the DE team and which is part of the sample application is based on which and how many, i.e. percentage, of the products use a component.
- Feature models are not used. Therefore the use of QFMs to derive the quantification of component usage and their impact is not validated by this case study.

9.5 Tool Support

In current practice spreadsheets are used for RBT in single system engineering and in the case study the risks and percentage of use where directly captured at component level and also do not require special tool support. However, when QFMs are used, as shown in Section 9.3, new requirements arise because spreadsheets do not provide a scalable solution.

Therefore, this section describes the modeling requirements and evaluates prior art to serve as a basis for further research.

9.5.1 Requirements

Tool support for RBT should support the process of creating the risk matrix. Here we focus on the steps which are specific for RBT in Software Product Line Engineering, see Section 9.3.6:

Estimate percentage of a feature in a product line: This estimation is done by the product managers of the different product groups as part of the commonality variability analysis. This can be based on a prediction of products that still need to be developed together with information from products that are already delivered to the market. These estimations need to be combined and captured in a tool. This tool should support the use of Quantified Feature Model, which will be defined in the next subsection.

Calculate percentage for each component: It should be able to model the dependencies between a feature model and the development artefacts, where the percentage of a development artefact, such as a component, can be derived from the features it implements.

9.5.2 Definition of quantified feature models

A QFM is a feature model in which each feature Fi is associated with a percentage P(Fi). This percentage is calculated as follows: We define the complete set of configurations as Ω. We then define: S(Fi). This is the set of configurations in which Fi is part of the configuration. We then define C(S(Fi)), as the number of elements, or Count, of the set S(Fi). Consequently C(Ω) is the total amount of configurations. The percentage is then $P(Fi) = C(S(Fi))/C(\Omega) * 100\%$.

Since features may have relations between them, e.g. parent child, requires relations and OR-groups, their associated percentages are also related. A QFM is valid when the feature model and the associated percentages are consistent. Some consistency rules are listed below (a more comprehensive set and their proofs is presented in Section 9.8:

- A *mandatory* feature Fm is present in every configuration so has a percentage of 100%.
- An *optional* feature Fo can have a percentage: $0 \leq P(Fo) \leq 100$
- *Parent-child relation*: The percentage of a child feature Fc cannot be greater than that of the parent feature Fp.: $P(Fc) \leq P(Fp)$.
- R*equires relation*: The percentage of the feature Fi cannot be higher than a feature Fj it requires: $Fi \rightarrow Fj$ then $P(Fi) \leq P(Fj)$.
- *Excludes relation*: When one feature Fi excludes another Fj, then the sum of the percentages of these features cannot be more than 100%: $Fi \rightarrow \neg Fj$ then $P(Fi) + P(Fj)) \leq 100\%$.

9.5.3 Extended feature models

The quantification of a feature can be captured by using extended feature models which are feature models that contain attributes and may include constraints among attributes and features 0.

Existing commercially available tools such as pure::variants [pure-variants 2006] and Gears [Biglever 2014] support this functionality. Figure 55 shows a feature model of the example from Figure 49 modeled in pure::variants.

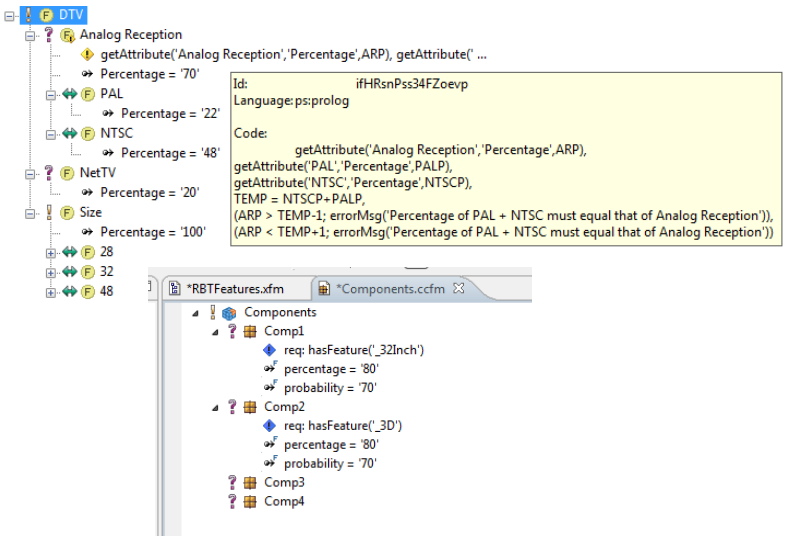

Figure 55 Screenshot of a feature a family model with quantification and consistency checking

The figure shows that each feature contains an attribute and a constraint that captures the consistency rule for the alternative child features PAL and NTSC of the optional feature Analog Reception. This consistency rule provides an error message when inconsistent percentages are entered.

The figure also shows a family model, which contains the reusable components with the percentage and impact derived from the feature model and the probability that is related to the component. As in the feature model, the family model also supports dependency relations with other development artefacts. Furthermore the percentage and impact for each component can be automatically derived from the feature models.

However, while existing tools support the basic functionality, they have two types of practical limitations.

- In situations where multiple features are involved complex consistency rules may occur:
  As an example consider the following scenarios:
  (1) Suppose we have two features (Fa, Fb) that exclude each other: $Fa \rightarrow \neg Fb$ and both features require another feature Fc: $Fa \rightarrow Fc$, $Fb \rightarrow Fc$. Then the percentage of

Fc is larger than the sum of the percentages of Fa and Fb together: P(Fc) ≥ P (Fa) + P (Fb)

(2) Suppose we have three features (Fa, Fb, Fc, Fd). Feature Fc and Fd are alternative features. Feature Fc requires Fa and excludes Fb; Feature Fd excludes Fa and requires Fb. As a consequence Fa and Fb exclude each other and we get the same consistency rule for Fa and Fb as described above.

To recognize and capture this type of relation in a model, becomes complex since the entire f model has to be analyzed.

- The percentages of features may not be independent as in the example of Section 9.3.2 where components C4 implements the combination of feature FC and FD. This would introduce additional percentages that capture these combinations.

Note that in the work of Benavides et. al [Benavides 2005] a feature commonality is defined as the percentage of products containing a feature, also using extended feature models. However, this definition and the method described to obtain the percentages are based on the percentage of a feature as part of the *possible* set of configurations, rather than the *expected* or *actually* created set.

9.5.4 Probabilistic feature models

An alternative to the use of extended feature models this section discusses Probabilistic Feature Models (PFMs) [Czarniecki 2008]. PFMs are feature models with soft constraints that express relations between features with a certain probability. PFMs can be used as belief measures where the probabilities can also be viewed as a relative frequency of seeing a particular outcome of an experiment in a large number of trials. In the case of QFMs these large number of trials represent all configurations and the percentage that is associated with each feature represent the probability that a particular feature is selected.

QFMs can be modeled with PFMs by adding a soft constraint to each dependency relation instead rather than using a feature attribute. For instance a parent with child features will have a soft constraint that describes the likelihood that a child feature is selected when the parent is selected. The percentages of each feature are automatically derived by the PFM through choice propagation, which in its initial state gives the percentage of each feature without a specific selection.

Since PFMs are based on belief measures they do not provide consistency checking for the hard dependency relations, such as requires or excludes relations, since belief measures introduce a chance, rather than a hard constraint. An advantage is that PFMs can be used to derive the percentages of features combinations, as show in the example in Section 9.3, i.e. feature FC and FD.

9.5.5 Further research on tool support

The current tools that are used for RBT, i.e. spreadsheet, are widely available and are easy to use, but do not provide a solution for modelling QFMs and the relation to development artifacts.

Current variability management tools support extended feature models including modelling of development artifacts, however complex relations between features and their

quantification require an intensive analysis of the feature model. Probabilistic feature models support a quantification using soft constraints, but do not provide the consistency checking.

Further research is needed to determine whether the current modelling approaches can be extended to support QFMs or that a new approach is required. Furthermore it needs to be evaluated what type of tools will be practically applicable in the industry. It should be realized that RBT is based on estimations rather than rigorous statistics and therefore a precise consistency checking may not be needed. Since our case study didn't include QFMs this needs to be further analyzed.

We believe that QFMs will not only benefit RBT, but may also benefit the commonality variability analysis. An estimate of feature usage can support the decision process on what features should be included in the product line and which should become application specific. Furthermore quantification can help in deciding which features should become optional and which should become mandatory, e.g. when used in a large percentage of the product line.

9.6 Comparison with Related Art

To our knowledge there is no prior art that discusses RBT for software product lines, and recent surveys confirm this [Lamancha 2013, Lee 2012, doCarmo 2012,]. We compare our research with methods to test efficiency, the selection of the test cases and test case generation.

9.6.1 Related art on test efficiency

Ganesan et.al [Ganesan 2007] discusses different test strategies, and uses a financial model to obtain the best strategy. In their work they didn't include risk as a factor and therefore didn't address individual quality requirements of software components.

Ensan et.al [Ensan 2011] reduce the test space by selecting the most desirable features and also capture this as part of a feature model. Their work is similar to ours by focusing on a set of features, but in their work they didn't address risks and quality requirements nor distinguished between testing during DE and AE.

9.6.2 Related art on selection of test cases

Several related art exists to limit the amount of test cases by reducing the combinatorial explosion, such as Kim et al. 0. While this analysis maybe used to limit the amount of test cases, this would still lead to a huge amount of test cases which remains unfeasible in most situations.

However this related art can be combined with the work in this paper. RBT can be used to preselect the development artifacts while the work on test case selection can further reduce the test space.

9.6.3 Related art on test case generation

A number of papers discuss the automation of test cases to handle the amount of variability, e.g. Segura et. al [Segura2010], Henard et.al [Henard 2012]. In the work of Devroey et. al [Devroey 2014] the likelihood of execution of a feature is used to generate test cases based on the usage model. In our work we have introduced the percentage of variants that use a feature rather than its usage. In RBT the usage is part of the impact of a feature. The percentage of use can be used to decide between testing during DE and AE while usage frequency is used to determine the required level of testing.

However, the main difference between this related art and our work, is that RBT does not generate test cases, but is the basis for test planning and to support the test strategy. Therefore our work and this related art can very well be combined as they are complementary contributions.

Another, related, difference is that our work is independent from the chosen technology and does not require a model or detailed knowledge of the behavior of a system. Furthermore, none of this related art discusses the risks as basis to prioritize tests.

Regarding the automated creation of test cases; it should be realized that in many cases visual inspection may be required to validate the outcome of a test [Veenendaal 2002]]and for these situations automated tests have limited value. For these situations, since verifying the outcome of such tests is very time consuming, reducing the amount of test cases while ensuring quality is crucial.

9.7 Conclusions and Further Research

In this paper we showed that the concept of risk based testing fits very well in the theory and practice of software product lines. We introduced a third dimension to risk based testing, i.e. the percentage of use of a feature in the variants of the product line. The three dimensions provide the criticality for an individual development artifact, such as a component or software unit, which is used to set the priorities for domain and application engineering.

The approach can be applied with different test strategies and addresses the advantages and disadvantages of these strategies.

We discussed the required tool support as a basis for further research. We identified the limitations of using spreadsheets, which is the current practice in risk based testing for single system engineering, the limitation of extended feature models and that of probabilistic feature models.

The case study showed that risk based testing is used in practice at various test levels of the test process, during domain as well as application engineering. In the case study the quantification of features and component was not formally captured as part of a feature model but spreadsheets were used to capture and visualize the risks that were identified at the level of components and software units.

We showed that our approach can be combined with related art, especially on the selection of test cases and test automation, since these contributions are complementary.

The contributions of this paper are three fold:

- It provides an approach for practitioners to apply risk based testing in software product line engineering.
- It provides academia a view on how practitioners set the priorities for test activities in product line engineering which enables further research on test efficiency.
- It provides the basis for further research, especially on the topic of quantified feature modelling.

We are certain that risk based testing will further find its way to the industry because of its intuitive nature and because it fits very well within existing practices. While our case study showed the application of risk based testing, it didn't validate the use of quantified feature models. Therefore we plan further case studies.

Acknowledgements

The authors thank Peter van Loon for his contributions to the case study of the Philips Healthcare Medical Workspot.

9.8 Appendix: Consistency Rules of Quantified Feature Models

This appendix contains the proofs of the consistency rules from Section 9.5.2.

A mandatory feature Fm is present in every configuration so has a percentage of 100%.
Proof: $S(Fm) = \Omega \Rightarrow C(S(Fm)) = C(\Omega) \Rightarrow P(Fm) = P(\Omega) = 100\%$.

An *optional* feature Fo can have a percentage: $0 \leq P(Fo) \leq 100$.
Proof: $\emptyset \subseteq S(Fo) \Rightarrow C(\emptyset) \leq C(S(Fo)) \Rightarrow 0 \leq P(Fi)$; $S(Fo) \subseteq \Omega \Rightarrow C(S(Fo)) \leq C(\Omega) \Rightarrow P(Fo) \leq P(\Omega) \Rightarrow P(Fo) \leq 100\%$

Parent-child relation: The percentage of a child feature Fc cannot be greater than that of the parent feature Fp.
Proof: In each configuration where the child feature Fc is present, then also the parent Fp is present $S(Fc) \subseteq S(Fp) \Rightarrow C(S(Fc)) \leq C(S(Fp)) \Rightarrow P(Fc) \leq P(Fp)$.

Requires relation: The percentage of the feature Fi cannot be higher than a feature Fj it requires: $Fi \rightarrow Fj$ then $P(Fi) \leq P(Fj)$.
Proof: This is the same as for a parent-child relation since a parent child relation is a special type of requires relation.

Excludes relation: When one feature Fi excludes another Fj, then the sum of the percentages of these features cannot be more than 100%: $Fi \rightarrow \neg Fj$ then $P(Fi) + P(Fj)) \leq 100\%$.
Proof: S(Fi) and S(Fj) are disjoint sets since an individual configuration cannot contain both features: $S(Fi) \cap S(Fj) = \emptyset$ Furthermore, the conjunction of the sets cannot be larger than Ω: $S(Fi) \cup S(Fi) \subseteq \Omega$ then follows:
$C(S(Fi)+S(Fj)) = C(S(Fi)) + C(S(Fj)) \leq C(\Omega) \Rightarrow P(Fi)+P(Fj) \leq 100\%$.

Exclusive OR group (also known as alternatives): In such a group the sum of the percentages of the child features, Fc1, Fc2,..Fcn, equals the percentage of the parent feature Fp: $\sum P(Fi)=P(Fp)$.
Proof: We have two sets of relations that govern this rule (1) Each of the alternative features exclude the others (2) Exactly one of the features is selected in a configuration. Following (1) S(Fi), for (all i,j =1..n) are disjoint sets: $S(Fi)\cap S(Fj)=\emptyset$ and following (2) The union of all the sets together exactly matches the set of the parent features: $\cup S(Fi)=S(Fp)$. We then get $\sum C(S(Fj))=C(S(Fp)) \Rightarrow \sum P(Fj)=P(Fp)$.

OR Groups with cardinality: In these OR groups the number of features selected is defined through a range with a lower- and upper-bound. For instance between 2 and 5 features must be selected from a group of e.g. 10 features. For instance 2 features are selected in one configuration and 4 features in another. There is a consistency rule for the lower-bound, Lb, and a consistency rule for the upper-bound, Ub.
Lower-bound consistency rule: The sum of the percentages of the n child features, Fc1, Fc2, ..Fcn, is larger than or equal to the percentage of the parent feature Fp multiplied by the lower-bound: $\sum P(Fci)\geq P(Fp)*Lb$.
Upper-bound consistency rule: The sum of the percentages of the n child features, Fc1, Fc2, ..Fcn, is smaller than or equal to the percentage of the parent feature Fp multiplied by the upper-bound: $\sum P(Fci)\leq P(Fp)*Ub$.
Note: In an exclusive OR group the lower-bound and upper-bound are both equal to 1 and fits this formula. Also the rule for optional features fits since there is only 1 feature in the group, the lower-bound equals 0 and upper-bound equals 1.
Proof: Note that now the sets S(Fci) are not necessarily disjoint since in a configuration more than 1 of the child features may be selected together. For our proof, first we make a few definitions:
Let Vj be a configuration in which Fp, the parent of the group, is selected. Hence $Vj\in S(Fp)$.We define T = C(S(Fp)), i.e. the total number of elements for which Fp is selected. Now we define S(Vj). This is the set of features Fci which are selected as part of the configuration Vj. Then C(S(Vj)) is the total number of elements of S(Vj).
From the cardinality of the group it follows that for each of the configuration Vj ,$\forall\ j\in(1..T)$ at least Lb features from the cardinality group are selected and at most Ub. Therefore: $Lb\leq C(S(Vj))\leq Ub, \forall\ j\in(1..T)$
Now we take the sum of the counts of the groups S(Vj): $\sum C(S(Vj))$. Since $Lb\leq C(S(Vj)), \forall j\in(1..T)$, It follows: $T*Lb=C(Fp)*Lb\leq \sum C(S(Vj))$. Since $C(S(Vj))\leq Ub, \forall\ j\in(1..T)$ It follows: $\sum C(S(Vj))\leq T*Ub = C(Fp)*Ub$. Because $P(Fp) = C(Fp)/ C(\Omega) *100\%$, it only remains to be proven that: $\sum C(S(Vj))=\sum C(S(Fci))$. Let's take a closer look at the set S(Vj). As defined this is the set of features Fc which are selected as part of the configuration Vj. So e.g. S(V2) contains the elements e.g. (Fc1, Fc3, Fn). The set S(Fci) is the set of configurations in which Fci is selected. E.g. S(Fc3) contains the elements e.g. (V1, V2, V5). Now each element of S(Vj) corresponds to exactly one element of S(Fci) and vice versa. In our example the member Fc3 of the set S(V2), corresponds to member V2 of S(Fc3), by definition. Therefore there is a bidirectional relation between all the members of the

combined group of sets S(Vj) ,∀ j∈(1..T) and that of the combined group of sets S(Fci) ,∀ i∈(1..n) and hence the number of elements of these two groups of sets is the same.

Feature cardinality: A feature that has cardinality can have multiple instances and each can have a percentage associated with it. So there is a percentage of a having one instance in the set of configurations, another percentage for the second instance etc. Features with cardinality can also have child features that consequently have multiple instances as well and have a percentage associated with it.